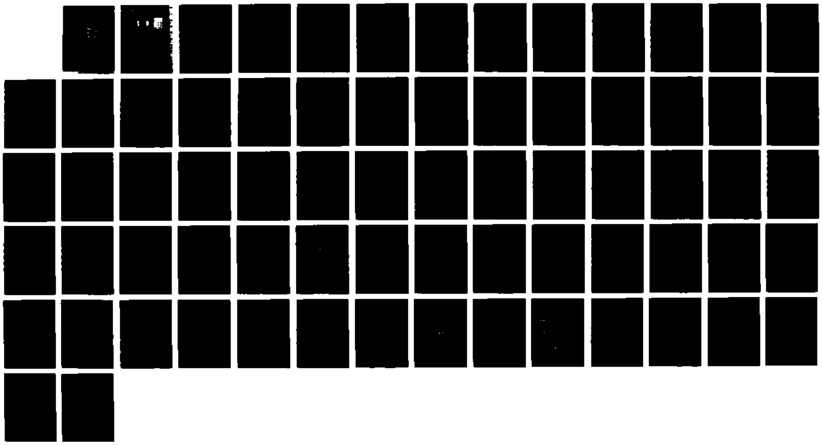
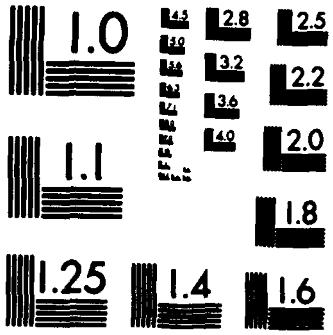
AD-A181 706 INTEGRATED INFORMATION SUPPORT SYSTEM (IISS) VOLUME 5 1/1
COMMON DATA MODEL S..(U) GENERAL ELECTRIC CO
SCHENECTADY NY PRODUCTION RESOURCES CONSU..
UNCLASSIFIED T LOOMIS ET AL. 01 NOV 85 PRM-620141200 F/G 12/5 NL





2

AFWAL-TR-86-4006
Volume V
Part 8

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 8 - NDML Programmer's Reference Manual

General Electric Company
Production Resources Consulting
One River Road
Schenectady, New York 12345

Final Report for Period 22 September 1980 - 31 July 1985

November 1985

MATERIALS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AFB, OH 45433-6533

This technical report has been reviewed and is approved for publication.

[illegible]

DAVID L. JUDSON, PROJECT MANAGER
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

5 Aug 1986
DATE

FOR THE COMMANDER:

[illegible]

GERALD C. SHUMAKER, BRANCH CHIEF
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

7 Aug 86
DATE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

A181706

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-86-4006 Vol V, Part 8
6a. NAME OF PERFORMING ORGANIZATION General Electric Company Production Resources Consulting	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION AFWAL/MLTC	
6c. ADDRESS (City, State and ZIP Code) 1 River Road Schenectady, NY 12345	7b. ADDRESS (City, State and ZIP Code) WPAFB, OH 45433-6533
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Materials Laboratory Air Force Systems Command, USAF	8b. OFFICE SYMBOL (If applicable) AFWAL/MLTC
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-80-C-5155	
8c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, Ohio 45433	10. SOURCE OF FUNDING NOS.
11. TITLE (Include Security Classification) (See Reverse)	

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
78011F	7500	62	01

12. PERSONAL AUTHOR(S)
Loomis,T. Loomis,M. Althoff,J. Apicella,M.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final Technical Report	22 Sept 1980 - 31 July 1985	1985 November	68

16. SUPPLEMENTARY NOTATION
ICAM Project Priority 6201 The computer software contained herein are theoretical and/or references that in no way reflect Air Force-owned or -developed computer software.

17. COSATI CODES

FIELD	GROUP	SUB. GR.
1308	0905	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Common Data Model Processor (CDMP) is a mechanism by which application programs can retrieve and update data without knowing where or how the data are stored. An application program poses requests to the CDMP, which processes those requests against the databases in which the relevant data are stored and then returns the results to the application program. The Neutral Data Definition Language (NDDL) is the means for posing requests to the CDMP. This manual explains the syntax and semantics of each NDML command. (To page vii)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
[✓ UNCLASSIFIED/UNLIMITED] SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
David L. Judson	513-255-6976	AFWAL/MLTC

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE. Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

11. Title

Integrated Information Support System (IISS)
Vol V - Common Data Model Subsystem
Part 8 - NDML Programmer's Reference Manual

A S D 86 1478
17 Jul 1986

PRM620141200
1 November 1985

PREFACE

This programmer's reference manual covers the work performed under Air Force Contract F33615-80-C-5155 (ICAM Project 6201). This contract is sponsored by the Materials Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Gerald C. Shumaker, ICAM Program Manager, Manufacturing Technology Division, through Project Manager, Mr. David Judson. The Prime Contractor was Production Resources Consulting of the General Electric Company, Schenectady, New York, under the direction of Mr. Alan Rubenstein. The General Electric Project Manager was Mr. Myron Hurlbut of Industrial Automation Systems Department, Albany, New York.

Certain work aimed at improving Test Bed Technology has been performed by other contracts with Project 6201 performing integrating functions. This work consisted of enhancements to Test Bed software and establishment and operation of Test Bed hardware and communications for developers and other users. Documentation relating to the Test Bed from all of these contractors and projects have been integrated under Project 6201 for publication and treatment as an integrated set of documents. The particular contributors to each document are noted on the Report Documentation Page (DD1473). A listing and description of the entire project documentation system and how they are related is contained in document FTR620100001, Project Overview.

The subcontractors and their contributing activities were as follows:

TASK 4.2

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Reviewer.
D. Appleton Company (DACOM)	Responsible for IDEF support, state-of-the-art literature search.
General Dynamics/ Ft. Worth	Responsible for factory view function and information models.

Subcontractors	Role
Illinois Institute of Technology	Responsible for factory view function research (IITRI) and information models of small and medium-size business.
North American Rockwell	Reviewer.
Northrop Corporation	Responsible for factory view function and information models.
Pritsker and Associates	Responsible for IDEF2 support.
SofTech	Responsible for IDEF0 support.

TASKS 4.3 - 4.9 (TEST BED)

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Responsible for consultation on applications of the technology and on IBM computer technology.
Computer Technology Associates (CTA)	Assisted in the areas of communications systems, system design and integration methodology, and design of the Network Transaction Manager.
Control Data Corporation (CDC)	Responsible for the Common Data Model (CDM) implementation and part of the CDM design (shared with DACOM).
D. Appleton Company (DACOM)	Responsible for the overall CDM Subsystem design integration and test plan, as well as part of the design of the CDM (shared with CDC). DACOM also developed the Integration Methodology and did the schema mappings for the Application Subsystems.

Subcontractors	Role
Digital Equipment Corporation (DEC)	Consulting and support of the performance testing and on DEC software and computer systems operation.
McDonnell Douglas Automation Company (McAuto)	Responsible for the support and enhancements to the Network Transaction Manager Subsystem during 1984/1985 period.
On-Line Software International (OSI)	Responsible for programming the Communications Subsystem on the IBM and for consulting on the IBM.
Rath and Strong Systems Products (RSSP) (In 1985 became McCormack & Dodge)	Responsible for assistance in the implementation and use of the MRP II package (PIOS) that they supplied.
SofTech, Inc.	Responsible for the design and implementation of the Network Transaction Manager (NTM) in 1981/1984 period.
Software Performance Engineering (SPE)	Responsible for directing the work on performance evaluation and analysis.
Structural Dynamics Research Corporation (SDRC)	Responsible for the User Interface and Virtual Terminal Interface Subsystems.

Other prime contractors under other projects who have contributed to Test Bed Technology, their contributing activities and responsible projects are as follows:

Contractors	ICAM Project	Contributing Activities
Boeing Military Aircraft Company (BMAC)	1701, 2201, 2202	Enhancements for IBM node use. Technology Transfer to Integrated Sheet Metal Center (ISMC).

Contractors	ICAM Project	Contributing Activities
Control Data Corporation (CDC)	1502, 1701	IISS enhancements to Common Data Model Processor (CDMP).
D. Appleton Company (DACOM)	1502	IISS enhancements to Integration Methodology.
General Electric	1502	Operation of the Test Bed and communications equipment.
Hughes Aircraft Company (HAC)	1701	Test Bed enhancements.
Structural Dynamics Research Corporation (SDRC)	1502, 1701, 1703	IISS enhancements to User Interface/Virtual Terminal Interface (UI/VTI).
Systran	1502	Test Bed enhancements. Operation of Test Bed.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION 1

INTRODUCTION

The Neutral Data Manipulation Language, hereafter NDML, was developed to provide access to the databases of the IISS Testbed. The NDML allows one to work with the heterogeneous distributed databases of the IISS Testbed as if they constituted a single relational database.

It has been designed to provide as much functionality as possible while attempting to be logical in application and convenient. The NDML is intended to be used by both data processing professionals and by manufacturing personnel who have little knowledge of database systems.

The NDML is a language similar to SQL (Sequel) and Quel, two well-known languages used to access relational databases. The utility of the method of access provided by the NDML is supported by extensive theory and practical tests of these relational languages.

The NDML is designed for use either as a stand-alone language or as embedded statements in the host languages of COBOL or FORTRAN. Currently, only embedded statements are supported and this manual applies only to embedded NDML. The NDML examples in the command descriptions of this manual neglect the embedding characters (*# for COBOL or C# for FORTRAN) for simplicity, but their use is shown in succeeding sections.

When stand-alone requests are supported, deviations from the embedded language will be as few as possible. The differences are due mainly to the requirement that a retrieved table be presented to host programs a row at a time, while the entire table can be presented in response to a stand-alone request from an interactive user.

The IISS Precompiler should process the application program containing embedded NDML statements before the host language (COBOL or FORTRAN) compiler is used. The host language compiler can be used first to debug host language statements, but the NDML precompile step must precede host-language compilation before executable object code is produced. The use of the Precompiler is described under "NDML Processing" in Section 4.

The important property of the NDML to keep in mind when using this manual is that the user perceives all "data" to be in the form of tables. Data within the database can be considered to be stored as tables even if containing only one row of values. Similarily, only tables can be retrieved from the database, even if the table consists of a single "row" with a single "column" (i.e., only a single value). This important property of relational databases allows the output of one retrieval command to be utilized as the input to another operation without worrying about the structure of '' e data. Furthermore, "chunks" of data can be retrieved and used without having to specify the structure of the data for each application and the size of the data chunk.

Tables are usually called "relations" and the terms table and relation will be used synonymously here. Similarly, rows of the table may be called "records" or "tuples" and columns may be called "data fields", "data items" or "attributes". An individual number or character string entry in the table will be called a "value".

Each of the *following sections on specific commands* begins with the syntax of the command. The syntax is presented using a method that is described at the beginning of Section 3; it is similar to the method used in the NDDL manual. The rigorous BNF description of the language is presented in an appendix. Following the syntax of the command, semantic notes point out conflicting commands and restrictions that are not supported by the system. These sections should provide sufficient information for the professional user to begin work. Moreover, the new user will find them an appropriate reference in the future when he or she has become familiar with NDML.

New users unfamiliar with SQL will want to consult tutorials and references on that language before using this guide.

References include:

Chamberlin, D.D. et. al., "Sequel 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM Journal of Research and Development. Vol 20, No. 6, Nov. 1976, pp. 560-575.

Date, C.J., A Guide to DB2. Addison-Wesley Publ. Co., 1984.

In addition, many commercial relational database systems offer SQL-like interface languages. The manuals for these languages are useful for becoming acquainted with the general syntax of SQL.

SECTION 2

SYSTEM OVERVIEW

The processing system is known as the Common Data Model Processor(CDMP). The CDMP provides the application programmer with important capabilities to:

- Request database accesses in a non-procedural data manipulation language (the NDML) that is independent of the data manipulation language (DML) of any particular data base management system.

- Request database access using a NDML that specifies accesses to a set of related records, rather than to individual records (i.e., using a relational DML).

- Request access to data that are distributed across multiple databases with a single NDML command, with minimum knowledge of data locations or distribution details.

Information about external schemas, the conceptual schema and internal schemas (including data locations) are provided by CDMP access to the Common Data Model (CDM) database. The CDM is a relational database of metadata pertaining to IISS. It is described by the CDM1 information model using IDEF1. The Precompiler parses the application program source code, identifying NDML commands. It applies external-schema to conceptual-schema and conceptual-schema to internal-schema transforms on the NDML command, thereby decomosing the NDML command into internal-schema, single atabase requests. These single database requests are each transformed into generic data manipulation language (DML) commands. Programs are generated from the generic DML commands which can access the specific databases to accomplish the request. These programs, referred to as Request Processors (RP), are stored at the appropriate host machines. The NDML commands in the application source program are replaced by host-language code which, when executed, activates the run-time request evaluation processes associated with the particular NDML command.

The Precompiler also generates a CS/ES Transformer program which will take the final results of the request, stored in a file as a table with external-schema structure, and convert the data values into the correct form for presentation. The CS/ES

Transformer also performs NDML function operations on the data.

Finally, the Precompiler generates a Join Query Graph and Result Field Table which are used by the Distributed Request Supervisor (DRS) during the run-time evaluation of the NDML request.

The DRS is responsible for coordination of the run-time activity associated with the evaluation of an NDML command. It is activiated by the application program, which sends it the names and locations of the query processors to activate along with run-time parameters which are to be sent to them. The results generated by the query processors are stored as files in the form of conceptual-schema relations on the host which executed the query process. Using the Join Query Graph, transmission cost information and data about intermediate results, the DRS determines the optimal strategy for combining the intermediate results of the NDML command. It issues the appropriate file transfer request, activates aggregators to perform unions, joins, and NOT IN SET operations, and activates the appropriate CS/ES Transformer program to transform the final results. Finally, the DRS notifies the application program that the request is completed, and sends it the name of the file which contains the results of the request.

The Aggregator is activated by the DRS. An instance of the Aggregator is executed for each union, join, and NOT IN SET operation performed. It is passed information describing the operation to be performed and the file names containing the operands of the operation. The DRS ensures that these files already exist on the host which is executing the particular Aggregator program. The Aggregator performs the requested operation and stores the results in a file whose name was specified by the DRS.

SECTION 3

NDML COMMANDS

The following conventions are used in the description of the NDML commands at the beginning of the following sections.

Notation

UPPER CASE WORDS denote keywords in the command

LOWER CASE WORDS denote user-defined words (entered in upper case)

{ } denotes that exactly one of the options within the braces must be selected by the user

...denotes repetition of the last element

[ ] denotes that the entry within the brackets is optional

| denotes an "or" relationship among the entries

_ denotes default option

Punctuation

The only punctuation allowed is:

(1) a "." to separate the table-label (.e., table alias) from the column-name. The table-label is used to match a column to a specific table in the list of tables referenced in the FROM clause,
(2) a ":" before the name of a host-language program variable, structure or file name that will receive returned values,
(3) a "," between entries in the list of tables in a FROM clause,
(4) a "," between subscripts to an array variable,
(5) a set of parentheses to enclose the column-list in an INSERT statement,
(6) a set of parentheses to enclose the object column of a function,
(7) a set of parentheses to enclose the values to be

inserted in an INSERT statement,
(8) a set of parentheses to enclose a program variable subscript list,
(9) a mandatory ":" or "loop-construct" (see the section: LOOP CONSTRUCT) at the end of the command.

Character Case

Only upper-case letters are recognized by the NDML Precompiler.

Word Length

Table labels are limited to 2 characters.

Table and column names are defined by the relational view in use.

3.1 Data Retrieval Commands

Syntax of Command

Data are retrieved from the database using the SELECT command. The command has the following syntax:

```
SELECT [WITH { EXCLUSIVE }  LOCK]
             { SHARED    }
             { NO        }

    [INTO { FILE 'file-name'          } ]
          { FILE ':variable-name'     }
          { STRUCTURE :variable-name  }
    [DISTINCT]
     { [table-label] ALL                                  }
     { expr-spec ...                                      }
     { :variable-name [(subscript, ...)] = expr-spec ...}
    FROM table-name [table-label], ...
    [WHERE predicate-spec [AND predicate-spec ...]]
    [ORDER BY column-spec [direction] ...]
     { ;              }
     { loop-construct }
```

where:

file-name and variable-name are defined in the host program,

table-label is a one- or two-character name,

table-name and column-name are defined for the relational view,

value is: a scalar variable | a quoted variable | a number in the host program

```
                 /           \
direction is:   | ASC         |
                | DESC        |
               <  ASCENDING    >
                | DESCENDING  |
                | UP          |
                | DOWN        |
                 \           /

               /                                            \
expr-spec is: |  column-spec                                 |
              |     /     \                                  |
              |    | AVG   |   ([DISTINCT] column-spec)      |
              |    | MEAN  |                                 |
             <    <  MAX    >                                 >
              |    | MIN   |                                 |
              |    | SUM   |                                 |
              |    | COUNT |                                 |
               \    \     /                                 /

column-spec is: { column-name              }
                { table-name.column-name   }
                { table-label.column-name  }

                     /                /    \                \
predicate-spec is:  | column-spec    |  =   |   value        |
                    |                |  !=  |                |
                    |               <   >    >               |
                    |                |  >=  |                |
                   <                 |  <   |                 >
                    |                |  <=  |                |
                    |                 \    /                 |
                    |                                        |
                    | column-spec    {  =  }   column-spec   |
                    |                { !=  }                 |
                     \                                      /
```

loop-construct is a list of program and (or) NDML statements enclosed in parentheses for the purpose of transferring retrieved values to program variables

<u>Comments</u>

(a) SELECT Keyword

The SELECT command is the only command used in NDML to retrieve data from the distributed database. This keyword must be the first word in the command.

(b) LOCK Phrase

A lock limits access to specific rows of tables while a transaction is being processed to prevent alteration of them during the transaction. A lock is owned by the transaction in which the SELECT statement occurs. An EXCLUSIVE lock denies access to all rows accessed by the transaction to all other processes. In addition, a request by any other transaction for any type of lock on the row will be caused to wait until the EXCLUSIVE lock is released. An EXCLUSIVE lock is normally used only when using an update command on a row, but might be needed in a SELECT request in a transaction to ensure that no other transaction can obtain a lock on the row. A transaction issuing a SELECT request may need to lock a selected row if it intends to update the row based on values retrieved earlier.

A SHARED lock also locks rows but allows other transactions to also lock a row. A SHARED lock is normally used in a SELECT command to ensure that a row is not changed by a contemporary MODIFY or DELETE transaction that must obtain an EXCLUSIVE lock to perform its function.

If no type of lock is specified in a lock-request, a NO lock is assumed unless the SELECT falls within an explicitly specified transaction. For example,

```
BEGIN TRANS
.......
COMMIT/UNDO;
```

causes a SHARED lock to be requested automatically.

The lock placed by a transaction depends on the implementation of locks in the particular database systems of the internal schema. The lock placed on the data in the internal schema by the local database manager usually locks either (1) only the accessed record or (2) the entire accessed table, depending on the local database. A "LOCK TABLE" command that will ensure that an entire table, rather than just a record, is locked is not provided in NDML at present. A user should assume that only each record accessed is locked.

(c) INTO Phrase and Variable Assignments

The data retrieved by a SELECT command can be either (1) placed into a file or program structure with the INTO phrase or (2) assigned to program variables using a variable-assignment construct.

The file name can be specified by using the keyword FILE and enclosing the file name in single or double quotes. If a colon is not the first character following the first quote, then the literal contents of the quoted character string will be taken to be the name of the file. If the first character following the first quote is a colon, then the rest of the character string will be taken to be the name of a program variable, the contents of which is the name of the file.

If the name of the file is quoted, the user should not supply the COBOL SELECT or FD layout for the file; the Precompiler will generate these and the internal file name. The file can be accessed by the application program by opening it, reading it into a working storage area and then closing it.

If the file name is contained in a program variable, the Precompiler will generate code to write it as an external file. To access the file, the appropriate code must be present to read an external file, including COBOL SELECT and FD statements.

The entire result of the SELECT will be placed in the file, one row per record, in the order normally produced by the SELECT command. A loop-construct should not be specified when the INTO phrase is used to place results in a file.

A structure is indicated to receive the retrieved data by the keyword STRUCTURE followed by a space, a colon and the program name of the structure. The defined data types for the fields in the structure must agree exactly with those for the corresponding column. For a structure target, only the first row returned will be placed in the target unless the application program contains code for a loop-construct following the SELECT command. The syntax of the loop-construct is described in a separate section below.

The alternative to the INTO phrase is to assign retrieved data directly to program variables. If a variable name is specified for a column, it is assumed that the defined data type for the variable agrees in type exactly with the type of the column in the external-schema view accessed by the NDML command. Subscripted variables can be used as variables to receive data. Only the first row returned will be placed in the target unless the application program contains code for a loop-construct following the SELECT command. The syntax of the loop-construct is described in a separate section below.

If neither a file, a structure, nor variables are specified to receive the result of the select command in embedded NDML in an application program, the Precompiler will reject the NDML SELECT statement. Thus, an assignment of retrieved columns to program variables or an INTO clause must be specified, but both cannot be specified. Also note that if ALL is specified for columns, an INTO phrase must be specified.

The following are examples of valid SELECT statements.

```
SELECT INTO FILE 'DEPT-FILE'
    D.DNO D.DNAME D.DLOC D.DSIZE
    FROM DEPT D
    ORDER BY D.DNO;
SELECT INTO STRUCTURE :DEPT-STRUCT
    D.DNO D.DNAME D.DLOC D.DSIZE
    FROM DEPT D
    ORDER BY D.DNO
    loop-construct
SELECT :DEPTNO = D.DNO :DEPTNAME = D.DNAME
    :DEPTLOC = D.DLOC :DEPTSIZE = D.DSIZE
    FROM DEPT D
    WHERE D.DLOC != 'LAX'
    loop-construct
```

(d) SELECT DISTINCT Phrase

The DISTINCT clause on a SELECT statement is used to specify that duplicate rows are to be removed prior to presentation of the results. Omitting the DISTINCT clause implies that duplicate rows are not removed.

The DISTINCT phrase refers to the entire set of selected columns following it. For example, SELECT DISTINCT ALL FROM T1 removes only those rows from T1 for which all column values are identical to those of another row in T1. The DISTINCT processing is applied to rows in their external-schema formats.

```
SELECT INTO FILE 'FILE-NAME' DISTINCT ALL
    FROM DEPT D
    WHERE D.LOC = 'LAX';
SELECT INTO FILE 'FILE-NAME' DISTINCT
    D.DNO D.DNAME D.LOC
    FROM DEPT D
    WHERE D.SIZE = 'LARGE';
```

(e) Restrictions on Column Specifications

Only columns from a table can be specified; quoted literal data to be duplicated in a column are not allowed, but can be introduced easily by the application programmer. Arithmetic expressions involving column data are also not supported; they can also be implemented easily directly in the application program. For example, the following commands are not supported:

```
SELECT INTO FILE 'FILE-NAME'
    EMP ' IS IN DEPARTMENT ' EMPDEPT
    FROM EMP;
SELECT INTO FILE 'FILE-NAME'
    'OVERHEAD IS ' 0.5 * AMOUNT
    FROM CONTRACTS;
```

The column specification ALL indicates all columns of the single table specified by the rest of the SELECT statement. The table can be derived from a single table indicated in the FROM clause, as (optionally) qualified by a WHERE clause. Alternatively, multiple tables can be specified in the FROM clause if a join operation is specified in a WHERE clause to combine them into the single table required. Finally, a specific set of columns can be indicated by using a table-label to specify a particular table in the FROM clause. For example, the following query is not supported:

```
SELECT INTO FILE 'FILE-NAME' ALL
    FROM TABLE1, TABLE2;
```

but the following queries are supported:

```
SELECT INTO FILE 'FILE-NAME' ALL
```

```
          FROM TABLE1, TABLE2
          WHERE TABLE1.CITY = TABLE2.CITY;
     SELECT INTO FILE 'FILE-NAME' E.ALL
          FROM EMP E, DEPT D
          WHERE E.DNO = D.DNO;
```

An important requirement that must be observed to use the ALL column specification is that an INTO phrase must indicate where to place the results of the SELECT because individual columns cannot be explicitly assigned to program variables in this syntax. The number of data fields and data types in the target structure or file must correspond to those of the columns, as discussed in the section above on the INTO phrase. The ALL specification is prone to error in embedded NDML because the number and order of columns can change if the table is reorganized. Note also that the ALL specification can refer to only one table. If more than one table is specified in the FROM clause, the appropriate table to which the ALL designation applies must be indicated using a table-label.

(f) Statistics Functions

Function expressions can be presented as the result of a SELECT statement only; they cannot be used in a WHERE or ORDER BY clause. These functions are used to specify that column statistics of AVG value, MAX value, MIN value, SUM value or COUNT of rows are to be produced. AVG and MEAN are synonyms.

The results of AVG (column) are the same as the results of SUM(column)/COUNT(column). All values are considered unless the optional DISTINCT phrase within the function clause is included, in which case duplicate values are removed prior to the function application.

SELECT cannot return both a table and the result of functions in a single statement. Thus, if one function is specified in an expr-spec, then all values to be retrieved must be the result of functions. It is permissible to retrieve the results of several functions, but the user should be aware that the values in the single row returned will not necessarily have any logical relationship.

MIN, MAX and COUNT can be applied to both numeric and string columns. AVG, MEAN and SUM can be applied only to numeric columns. Functions are applied to columns in their external-schema formats. Statistic functions ignore nulls in the data. For the empty set, COUNT returns zero and other functions return an undefined result; the existence of the empty set for non-COUNT functions results in a condition code set in NDML-STATUS, as discussed below.

The ORDER BY clause should not be used when functions are specified because unnecessary processing will be performed (the system may not allow the clause to be specified). Specification of function DISTINCT before MIN or MAX is ignored. Functions cannot be used in a WHERE clause because the result of a function is a property of a group of rows rather than of each row. A SELECT DISTINCT specification should not be used with functions because it causes unnecessary processing.

The formats of function results in COBOL are the following: AVG, MEAN and SUM: S9(9)V9(9); COUNT: S9(9). The formats of function results in FORTRAN are the following: AVG, MEAN and SUM: F20.9; COUNT: I10. The number of rows returned by the request is contained in the variable NDML-COUNT generated into the application program by the Precompiler; obviously, it will al ways have a value of one for function requests. The variable NDML-STATUS (or NSTATS in FORTRAN) generated into the applications program contains a code that indicates the success or failure of the request. An all zero code indicates successful completion; another code indicates an error. If a function operates on an empty column, a result may be returned that is not really valid (for example, SUM will return 0.). The NDML-STATUS flag should be checked by the application program before using the result returned by a function. A full list of error codes is not available at this writing.

User-defined functions and explicit arithmetic functions (e.g., WEIGHT * 2.2) are not supported in this release.

```
SELECT INTO FILE 'FILE-NAME'
    AVG(P.LEAD-TIME) MIN(P.LEAD-TIME) MAX(P.LEAD-TIME)
    FROM PART P
    WHERE P.SIZE > 100;
SELECT INTO FILE 'FILE-NAME' COUNT(D.DNAME)
    FROM DEPT D, EMP E
    WHERE E.DNO = D.DNO;
SELECT INTO FILE 'FILE-NAME'
    MIN(SE.SALARY) MIN(HE.RATE)
    FROM SALARIED-EMP SE, HOURLY-EMP HE;
SELECT INTO FILE 'FILE-NAME' COUNT(DISTINCT E.JOB)
    FROM EMP E
    WHERE E.DNO = 10;
SELECT INTO FILE 'FILE-NAME' COUNT(DISTINCT D.LOC)
```

```
FROM DEPT D, NEWDEPT N
WHERE D.NAME = N.DNAME;
```

(g) FROM Clause

Table labels or table names may or may not be required by the syntax of the particular request. If two or more tables are specified in the table-list, it is a good idea to be concise and use table labels or table names to designate columns. When a table is joined with itself, it is necessary to use table labels to distinguish columns.

(h) WHERE Clause

The WHERE clause is used to limit the information returned from one or more tables. If the WHERE clause is not specified, all rows from the first table indicated in the table-list are returned (additional table names are ignored).

Only column-predicate or join-predicate comparisons are allowed in WHERE clauses. The column-predicate compares the value of a column with a single specific value indicated by the contents of a scalar program variable, a literal string in quotes, or a number. Either the column name or value can be the first object of the comparison (only the case in which the value is second is shown in the syntax above). AND clauses can be used to specify multiple qualifications on the table selected; however, an "OR" capability is not implemented at this time. The comparison operator (bool-op) includes most common operations but does not include an "IN" comparison that would allow a column to be compared with many values.

The absence of an "IN" bool-op and "OR" clauses restricts the ability to specify alternative qualifications on the selected table. In this release, multiple SELECT requests must be issued to retrieve all the information if a column can have more than one value of interest. The NDML command can easily be placed within a user-defined program loop within an application program. Consequently, subqueries, in which the comparison values are returned by another SELECT request, are not supported because more than one value can be returned by the subquery. Other possible comparison operators not supported include EXISTS, ALL and ANY.

Note that changing the contents of a program variable within the "loop-construct" of the SELECT command will have no effect on the result because the query has already been executed before the loop-construct is activated. A loop-construct is used only to transfer data from a completed SELECT query to program variables or to a structure. The loop-construct is described in detail in the section "LOOP CONSTRUCT" below.

Supported Query:

```
SELECT INTO FILE 'FILE-NAME' DNO DNAME
    FROM EMP
    WHERE DTYPE = 'SALES'
        AND DLOC = 'SOUTH';
```

Unsupported Queries:

```
SELECT INTO FILE 'FILE-NAME' DNO DNAME
    FROM DEPT
    WHERE DNO IN
          (SELECT DNO
               FROM LOCATION
               WHERE DEPTLOC = 'LA');

SELECT INTO FILE 'FILE-NAME' DNO DNAME
    FROM DEPT
    WHERE DSIZE = 'SMALL'
        OR DSIZE = 'MEDIUM';
```

The join-predicate comparison allows only the equijoin (=) and NOT IN SET (!=) operations; the operators <, <=, > and => are not implemented. The join fields compared in a join or NOT IN SET operation need not have identical data types in the user's (external) view of the table, except that numeric data must be compared with numeric data and character strings with character strings. All data will actually be compared in conceptual-schema format.

The equijoin connects a row from each of two tables to form one row in the result table if the values in the specified columns in the tables are identical. Duplicate rows will be returned if duplicate rows exist in either table. Rows for which a match are not found are not included in the result table.

The NOT IN SET operation is a selection procedure that eliminates each row from the first table for which the value in the specified column is found in any row in the specified column in the second table. In other words, the NOT IN SET operator is used to select all rows of a table where the value of a certain column is not equal to any value in a given column in another table. The order of the columns in the NOT IN SET predicate is significant. For example the following two requests yield different results:

```
        SELECT INTO FILE 'FILE-NAME' D.DNO D.DNAME
            FROM DEPT D, EMP E
            WHERE D.DNO != E.DNO;
         SELECT INTO FILE 'FILE-NAME' E.DNO E.NAME
             FROM DEPT D, EMP E
             WHERE E.DNO != D.DNO;
```

If the following data are found,

D.DNO	E.DNO
1	2
2	3
4	5
5	6
7	8
8	9

```
the result of first example is,       D.DNO
                                          1
                                          4
                                          7

and the result of second example is:  E.DNO
                                          3
                                          6
                                          9
```

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The user can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

(i) ORDER BY Clause

The ORDER by clause is used to specify the sequencing rules for presentation of the results of a SELECT operation. Omitting the ORDER by clause on a SELECT statement implies that the rows of the result table are presented in a system-determined order.

The columns in the order-spec-list control the sorting of result rows in major-to-minor order. If the direction phrase is omitted for a column, then ASC (ascending) is assumed. The columns of an order-spec-list need not all have the same accompanying direction. Also, the columns need not appear in the column-list of the SELECT phrase.

Sorting is done on the columns in their external-schema formats and will be done on the machine running the application program. The order of the sorted result will depend on the storage code used by the computer running the applications program. Thus, the result of the same program can differ if it is precompiled and run on different machines. Note that ASC, ASCENDING and UP are equivalent and that DESC, DESCENDING and DOWN are equivalent.

```
SELECT INTO FILE 'FILE-NAME' E.NAME E.DEPT E.PHONE
    FROM EMP E
    WHERE E.JOBCODE > 50
    ORDER BY E.NAME;
SELECT INTO FILE 'FILE-NAME' PART# SIZE
    FROM PART
    ORDER BY SIZE DESCENDING;
SELECT INTO FILE 'FILE-NAME' D.DEPT# D.LOC D.CITY
    FROM DEPT D
    ORDER BY D.CITY ASC D.LOC DESC D.SIZE ASC;
```

(j) Nulls

The effect of nulls in data are not sufficiently established at this writing to describe the result for specific operations. Because nulls are implemented in many ways in the internal-schema databases, it is not advisable to depend upon a particular response of the system to null data.

(k) Grouping Clauses

This release does not support GROUP BY and HAVING clauses to determine aggregate properties of multiple rows of a table. These operations must be performed by the application program.

3.2 DELETE Command

Syntax of Command

The DELETE command removes rows from an external-schema table. The DELETE command has the following syntax:

```
DELETE FROM table-name [table-label]
   [USING table-name [table-label], ...
   WHERE { ALL                                      }   ;
         { predicate-spec [AND predicate-spec ...] }
```

where:

table-label is a one- or two-character name,

table-name and column-name are defined for the relational view,

```
column-spec is: { column-name             }
                { table-name.column-name  }
                { table-label.column-name }
```

```
                      /              /   \                   \
predicate-spec is:   | column-spec  |  =  | value             |
                     |              | !=  |                   |
                     |              |  >  |                   |
                    <              <  >=   >                   >
                     |              |  <  |                   |
                     |              | <=  |                   |
                     |               \   /                    |
                     | column-spec  {  =  }  column-spec      |
                     |              { !=  }                   |
                      \                                      /
```

Comments

(a) Locking

A DELETE command inside a transaction usually places a "key lock" on deleted rows until a COMMIT command is encountered. This lock ensures that another process cannot insert a row with the key of the deleted row until the DELETE action has been finalized by a COMMIT command. A DELETE command outside of a transaction is usually committed immediately and no lock is used. Actual lock mechanisms depend on the internal-schema databases.

(b) USING Clause

The USING clause specifies tables that are accessed by the WHERE clause to qualify the request. These tables are not to have rows deleted from them. To be meaningful, tables indicated in the USING clause must be related to the table on which the DELETE command acts by a join-predicate.

(c) WHERE Clause

The WHERE clause is used to specify which rows qualify to be deleted. The WHERE clause is mandatory and the Precompiler will reject the request if it is not present. If all the rows of a table are to be deleted, the WHERE ALL clause should be used. For selective qualification of rows, the WHERE clause has the same power of expression as it does in a SELECT statement. Note that this release supports only the AND connector between qualifications in a WHERE clause.

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The use can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

(d) Mapping Restrictions

The external-schema table (the table the user views) must map to one complete conceptual-schema entity class. This means that a request to DELETE a row in a table in the user's view can be rejected by the system because other information that the user is not (necessarily) aware of would also have to be deleted in the conceptual-schema representation of the database. Thus, it may be necessary to determine the conceptual-schema structure and mapping to external views to formulate a correct DELETE command to explicitly delete all the columns of a row in the conceptual schema. In general, an effort should be made on the part of the CDM designers and administrator to make this mapping as simple as possible while still supporting the variety of external views needed by users.

The entity class (in the conceptual schema) may map to just part (or all) of one or more record types in the actual database (in the internal schema). If just part of a record type is mapped to, that deleted part is filled will null-values and the remainder is left as is; the designation of null values depends on the particular database manager.

(e) Integrity Constraints

A request to delete a conceptual-schema entity that has dependent entities will be rejected at runtime. Those dependent entities cannot be left hanging; their existence depends on the existence of the independent entity.

A future release may support DELETE WITH CASCADE, which will delete any dependent entities associated with the specified entity.

(f) Null Values

The specification of internal-schema null-values is DBMS dependent. When the DBMS does not support nulls (as in the case of TOTAL), all hyphens will be used in alphanumeric fields, negative zero will be used in signed numeric fields, and zero will be used in unsigned numeric fields.

(g) Examples:

```
DELETE FROM OFFER F
     WHERE F.STATUS = 'EXPIRED';

DELETE FROM OFFER
     WHERE ALL;

DELETE FROM OFFER F
     WHERE F.STATUS = 'OLD'
         AND F.DATE < :CUT-DATE
         AND F.TYPE != 'RETRO';

DELETE FROM OFFER F
     USING PRODUCT PR
     WHERE F.TYPE = PR.TYPE
         AND PR.CLASS = 'REPLACED';
```

3.3 <u>INSERT Command</u>

<u>Syntax of Command</u>

The INSERT command adds rows to an external-schema table.

The INSERT command has the following syntax:

```
        INSERT INTO table-name (column-spec ...)

                 /                                     \
           VALUES | FROM { FILE 'file-name'            } |  ;
                <        { FILE ':variable-name'       }   >
                 |       {  STRUCTURE :variable-name }  |
                 | value ...                             |
                 \                                     /
```

where:

file-name and variable-name are defined in the host program,

table-name and column-name are defined for the relational view,

table-label is a one- or two-character name,

value is: a scalar variable | a quoted variable | a number in the host program

```
column-spec is: { column-name              }
                { table-name.column-name   }
                { table-label.column-name  }
```

<u>Comments</u>

(a) Locking

An INSERT command issued inside a transaction usually places an EXCLUSIVE lock automatically (on rows or on tables, depending on the particular internal-schema database managers) until a COMMIT command is encountered. An INSERT command outside of a transaction is usually committed immediately and no lock is used.

(b) Specified Columns

The columns of the table are specified in the column-list. Values are supplied either from an external file, in which case many rows may be created, or from a source-list or data structure, in which case one row is created for each set of values. The values are related to columns in the column-list by their respective orders of appearance. The columns in the column-list need not be specified in the same order as the columns in the external-schema table were initially described to the system.

(c) File Input

If the values to be inserted are taken from a file, then multiple records can be inserted. The specification of file input causes an implicit loop to be generated that repeatedly executes the INSERT command until the file is empty. The file is read as string input. Each row is a record; the end of the rows is marked by the end-of-file. There are no delimiters between fields. It is assumed that the record format matches the format of the column list. The file-name is a logical file-name which should be related to a physical file through the system's job control language. The input file must be defined in the application program (by COBOL SELECT and FD statements).

(d) Structure Input

The format of a data structure must match the format of the column list. It is assumed that the data type of structure fields exactly match that of the corresponding table columns in the external-schema format. Only one row can be inserted by this method without explicitly placing the NDML command within a program loop.

(e) Value and Variable Input

A source list enclosed in parentheses can contain values and/or program variables for input. Multiple source-lists can be specified to cause an implicit loop to be generated that executes the INSERT command once for each source list. The data types of values explicitly given must agree with the data type of target columns. In this release, values cannot be calculated by an arithmetic expression within the INSERT statement.

(f) Mapping Restrictions

The external-schema table (the table the user views) must map to one complete conceptual-schema entity class. This means that a request to INSERT a row in a table in the user's view can be rejected by the system. Thus, it may be necessary to determine the conceptual-schema structure and mapping to external views to formulate a correct INSERT command to explicitly insert all the columns of a row in the conceptual schema. In general, an effort should be made on the part of the CDM designers and administrator to make this mapping as simple as possible while still supporting the variety of external views needed by users.

The entity class (conceptual schema) may map to just part (or all) of one or more internal-schema (actual databases) record types. If just part of a record type is mapped to, that part not inserted is filled will null-values. Moreover, if a record type in the internal database maps to two conceptual-schema entity classes, inserting in one conceptual entity, followed by the other, will result in two partial record instances in the internal database, rather than one complete instance; the Precompiler does not view this result as incorrect and will not issue a rejection or warning.

(g) Integrity Constraints

A request to insert a conceptual-schema entity that is dependent in a relation class but for which no independent entity exists will be rejected at runtime. A dependent entity cannot exist without its associated independent entities, one for each relation class in which it is dependent.

A request to insert a conceptual-schema entity with key value equal to that of an entity already in the database will be rejected at runtime. Key values must be unique.

(h) Null Values

The specification of internal-schema null-values is DBMS dependent. When the DBMS does not support nulls (as in the case of TOTAL), all hyphens will be used in alphanumeric fields, negative zero will be used in signed numeric fields, and zero will be used in unsigned numeric fields.

(i) Examples:

```
INSERT INTO DEPT
     (DNO DNAME DLOC DSIZE)
     VALUES FROM DEPT-FILE;

INSERT INTO DEPT
     (DNO DNAME DLOC DSIZE)
     VALUES (12 'ENGR' 'B1' 'SMALL');

INSERT INTO DEPT
     (DNO DNAME DLOC DSIZE)
     VALUES (12 'ENGR' 'B1' 'SMALL')
```

```
              (40 'CUST' 'F4' 'SMALL')
              (36 'SW' 'G2' 'LARGE');

      INSERT INTO DEPT
           (DNO DNAME DLOC DSIZE)
           VALUES (:DEPT-NUM :DEPT-NAME 'B1' :DEPT-SIZE);

      INSERT INTO DEPT
           (DNO DNAME DLOC DSIZE)
           VALUES FROM STRUCTURE :DEPT-REC;
```

where DEPT-REC has the structure:

```
           01  DEPT-REC.
               03  DEPT-NUM            PIC 99.
               03  DEPT-NAME           PIC X(4).
               03  DEPT-LOC            PIC XX.
               03  DEPT-SIZE           PIC X(5).
```

3.4 MODIFY Command

Syntax of Command

The MODIFY command changes values in an external-schema table. The MODIFY command has the following syntax:

```
      MODIFY table-name [table-label]
         [USING table-name [table-label], ...]
         SET column-spec = value ...
         WHERE { ALL                                      }   ;
               { predicate-spec [AND predicate-spec ...] }
```

where:

table-label is a one- or two-character name,

table-name and column-name are defined for the relational view,

value is: a scalar variable | a quoted variable | a number in the host program

```
      column-spec is: { column-name                }
                      { table-name.column-name    }
                      { table-label.column-name   }
```

```
                     /                 /   \               \
predicate-spec is:  | column-spec |   =   |  value          |
                    |             |  !=   |                 |
                    |            <    >    >                |
                   <              |  >=   |                  >
                    |             |   <   |                 |
                    |             |  <=   |                 |
                    |              \     /                  |
                    | column-spec  {  =  }  column-spec     |
                    |              { !=  }                  |
                     \                                     /
```

The columns to be changed and the values to be entered must be explicitly specified in the SET clause; values cannot be read from a structure or file.

<u>Comments</u>

(a) Integrity Constraints and Mapping Restrictions

Three specific integrity constraints are enforced by the system. First, the MODIFY command cannot be used to change the values of a column that corresponds to the key class of an entity class in the conceptual schema. Thus, some requests that have an apparently correct syntax might be rejected. To modify a key class, it is necessary to first DELETE and then INSERT the entity. Second, referential integrity is enforced. If a foreign key class is to be modified, there must exist a parent for the new key. Third, it is not permissible to change just part of a foreign key class; the entire foreign key must be changed.

Some columns cannot be modified alone because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The user can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

(b) Locking

A MODIFY command within a transaction usually places an EXCLUSIVE lock automatically (on rows or on tables accessed, depending on the particular internal-schema database managers) until a COMMIT command is encountered. A MODIFY command issued outside of a transaction usually commits the result immediately. The specific lock used is determined by the particular internal-schema database manager.

(c) USING Clause

The USING clause specifies tables that are accessed by the WHERE clause to qualify the request. These tables need not necessarily include the one that is being modified. To be meaningful, tables indicated in the USING clause must be related to the table on which the MODIFY command acts by a join-predicate.

(d) SET Clause

The SET clause specifies the new values that are to be given to values in designated columns. The new value can be contained in a program variable or be given explicitly. In this release, new values cannot be calculated by arithmetic expressions in the MODIFY command, nor can they be contained in a structure or file.

(e) WHERE Clause

The WHERE clause is mandatory. The WHERE clause is used to specify which rows qualify to be changed. If all the rows of a table are to be modified, then the WHERE ALL clause should be used. For selective qualification of rows, the WHERE clause has the same power of expression as it does in a SELECT statement. Note that this release of the system supports only the AND connector between phrases of a predicate. If the WHERE clause is not included in a MODIFY statement, the Precompiler will reject the statement and issue an error code.

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The user can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

(f) Examples:

```
MODIFY OFFER F
     SET F.STATUS = 'EXPIRED'
     WHERE F.DATE < :CUTDATE;

MODIFY OFFER F
     SET F.RESPONSIBLE-DEPT = 'BENEFITS'
     WHERE ALL;
MODIFY DEPT D
     USING EMPLOYEE EMP
     SET D.STATUS = 'INACTIVE'
     WHERE D.DNO != EMP.DNO;

MODIFY DEPT D
     SET D.STATUS = 'INACTIVE'
          D.LOC = 'INACTIVE'
          D.RESPONSIBLE-MNGR = :MNGR-INPUT
     WHERE D.DNO = :DEPT-NO-INPUT;
```

3.5 Transaction Commands

3.5.1 BEGIN TRANSACTION Command

The BEGIN TRANSACTION command indicates the start of one or a group of NDML commands that must be completed successfully as a unit in order to maintain the integrity of the database system. All automatic locks issued (for SELECT, INSERT, DELETE and MODIFY commands) and an explicit EXCLUSIVE lock placed by a SELECT command refer to this transaction. If locks exist from prior commands for an open transaction that have not been removed by a preceding commit-command or rollback-command, the BEGIN TRANSACTION command will issue a rollback-command to undo any uncommitted previous commands.

A transaction ends at the next UNDO, ROLLBACK or COMMIT statement. Transactions cannot be nested.

3.5.2 UNDO and ROLLBACK Commands

These NDML commands cause the system to undo any actions accomplished since the last BEGIN TRANSACTION command. The databases will be returned to their previous states.

3.5.3 COMMIT Command

The COMMIT command causes all actions accomplished since the last BEGIN TRANSACTION command to become permanent and all existing locks on records for this transaction to be removed. The following is an example use of the COMMIT command:

```
*#    BEGIN TRANSACTION;
*#    MODIFY OFFER F
*#        SET F.RESPONSIBLE-DEPT = 'BENEFITS'
*#        WHERE ALL;
      IF NDML-STATUS = 'ERROR'
*#        ROLLBACK;
      ELSE
*#        COMMIT;
```

3.6 Embedding NDML in COBOL

COBOL compilers do not know the meaning of the NDML commands. Therefore, a COBOL application program that contains NDML statements must be precompiled by the IISS Precompiler. The Precompiler substitutes in-line COBOL code into the application program in place of the NDML statements. The substituted code provides the mechanisms necessary to send messages to the Distributed Request Supervisor, make the contents of program variables available to the system, open and read returned files, etc.

In order to allow the Precompiler to distinguish the NDML statements from COBOL statements, each line of NDML code must contain a "*#" in columns 7 and 8; the rest of the NDML statement must begin in column 12 or greater. All NDML lines will look like comment lines to the COBOL compiler and the COBOL compiler can be used to test COBOL code before precompilation, with two exceptions, without removing the NDML statements. One exception is that a "." may be needed within a loop construct after a BREAK, EXIT, NEXT or CONTINUE statement that may not make sense to the COBOL compiler before the program has been precompiled; this results in a simple error message that can be ignored. The second exception is that any references to the variables NDML-COUNT or NDML-STATUS will result in a compile error because these variables are generated into the program by the Precompiler. Examples are found in the Appendix.

The following procedure is recommended for programming an application program containing embedded NDML.

1. Compile the program with the COBOL or FORTRAN compiler using explicit test values for program variables that will be retrieved by NDML commands. Locate and fix all

COBOL syntax and processing errors.

2. Precompile the program to identify all syntax errors in the NDML; fix all errors.

3. Compile the output of the Precompiler with the COBOL compiler and perform a number of link and install steps as described in PART II of this manual. Be sure to test the resulting program before routine use.

3.7 Embedding NDML in FORTRAN

FORTRAN compilers do not know the meaning of the NDML commands. Therefore, a FORTRAN application program that contains NDML statements must be precompiled by the IISS Precompiler. The Precompiler substitutes in-line code FORTRAN into the application program in place of the NDML statements. The subroutines provide the mechanisms necessary to send messages to the Distributed Request Supervisor, make the contents of program variables available to the system, open and read returned files, etc.

In order to allow the Precompiler to distinguish the NDML statements from FORTRAN statements, each line of NDML code must contain a "C#" in the first two columns; the rest of the NDML statement must begin in column 7 or greater. All NDML lines will look like comment lines to the FORTRAN compiler and the FORTRAN compiler can be used to test FORTRAN code, with one exception, without removing the NDML statements. The exception is that any references to NCOUNT or NSTATS (the equivalent of NDML-COUNT AND NDML-STATUS in COBOL) variables in the FORTRAN program will result in an error because these variables are generated into the program by the Precompiler.

The following procedure is recommended for programming an application program containing embedded NDML.

1. Compile the program with the FORTRAN compiler using explicit test values for program variables that will be retrieved by NDML commands. Locate and fix all FORTRAN syntax and processing errors.

2. Precompile the program to identify all syntax errors in the NDML; fix all errors.

3. Compile the output of the Precompiler with the FORTRAN compiler and perform a number of link and install steps

as described in PART II of this manual. Test the resulting object code.

3.8 Loop Construct

3.8.1 When a Loop Construct Is Needed

The host language compiler expects that all input and output in an application program be done a record at a time. In contrast, a single NDML SELECT command can return many records. The loop construct is provided to allow NDML to interact with the application program one record at a time.

A loop construct is necessary for assignment of multiple returned values to program variables or to a structure, even if the variables or structure fields are vectors. The major reason that implicit looping is not generated is that there is no way to determine the number of records to be returned during the precompile step; therefore, the programmer should test the number of records returned within an NDML loop construct to ensure that storage dimensions of the variables are not exceeded during execution.

It is not necessary to use a loop construct if only the first record returned is to be used. For example, a loop construct will never be necessary when functions are specified in a SELECT because only one row is returned. Specification is used because looping is implicit (the file is assumed to be capable of growing to hold all output).

Note that the loop executes after the SELECT retrieval is complete. Therefore, changing values in the WHERE or ORDER BY clauses within the loop will have no affect on the result.

3.8.2 Number and Type of Data Fields

The programmer must take care to specify the program variables or file names that are to receive the values returned by the SELECT statement. Note that the command

```
SELECT ALL
..................
```

is especially risky because the number of the retrieved columns is not explicit and their assignment to data fields in a structure or to a file is susceptible to error (i.e., assignment to individual program variables is syntactically not possible).

3.8.3 Syntax

A loop must immediately follow a SELECT command. If a loop construct follows, do not end the SELECT command with a ";"; the end of the NDML procedure is indicated by the closing bracket. The start of the loop is indicated by "{" and the end by "}", both of which are embedded NDML statements and must be preceded on the line in the application program by appropriate NDML designation characters. The body of the loop can contain both host-language statements and embedded NDML commands.

It is permissible to include NDML statements within loop constructs for a SELECT statement. A transaction defined by a BEGIN TRANSACTION statement must either enclose the entire SELECT statement and associated loop construct or must be contained within the loop construct. An example of the latter is given below.

Two important restrictions on the use of loop constructs are the following. Programmers should not attempt to exit a loop by using a host language GOTO or equivalent statement. The result of such a jump is undefined. Secondly, the NDML commands SELECT, INSERT, DELETE and MODIFY should not appear within a host-language "IF" statement because the Precompiler will not be able to guarantee the integrity of the logic path. The NDML statements listed below are provided to control the processing of loops. (The NDML commands COMMIT, UNDO and ROLLBACK can also be placed within a host-language IF statement).

3.8.4 NDML Loop Control Statements

(a) CONTINUE or NEXT
This statement causes the current iteration of the loop to terminate and the next iteration to be generated. The NDML statement CONTINUE should not be confused with the FORTRAN statement.

(b) BREAK or EXIT
This statement causes the loop to be terminated and control to be passed to the program statement following the end of the loop.

3.8.5 Evaluation

The following actions are taken by the system to evaluate an embedded NDML SELECT statement:

1. The system evaluates the query and stores the resulting rows in a result file. If a file name has been specified by the programmer in an INTO phrase to receive the results, the result file is given the specified name and the command is finished. Otherwise, proceed.

2. The code within the loop specified in the SELECT command is executed, once for each row generated by the query. Values are moved to the program variables or structure fields specified to receive them. It is necessary that the host language code either move those values to safe storage or specify new variables (for example, new indices of array variables) for each execution of the loop if more than one row is returned. The host language code should also test the number of loops to ensure that the allocated storage for returned information is not exceeded.

The following example illustrates how program variables that receive information from a SELECT statement can be manipulated in a loop construct (this and the following examples are COBOL). Note that the braces should be on a separate line without following code.

```
*#   SELECT :PART-NUMBER = P.PARTNO,
*#       :PART-NAME = P.NAME
*#       FROM PARTS P
*#   {
         DISPLAY PART-NUMBER, PART-NAME
         COMPUTE NUMBER-OF-PARTS =
            NUMBER-OF-PARTS + 1.
*#   }
```

The following example shows how a COBOL variable can be used in the WHERE clause and how the CONTINUE statement can be used. Parts with a null part name are skipped. Otherwise, counters are incremented depending on the value of the work number.

```
*#  SELECT :PART-NAME = P.NAME, :WORK-NO = P.WORKNO
*#      FROM PARTS P
*#      WHERE P.SIZE = 'SMALL'

*#         AND P.PARTTYPE = :PART-TYPE
*#  {
```

```
             IF PART-NAME = SPACES
*#              CONTINUE
             .
             IF WORK-NO < BREAK-POINT
                ADD 1 TO ODD-LOT-COUNT
             ELSE
                ADD 1 TO REGULAR-LOT COUNT.
*#    }
```

The following example shows the inclusion of a transaction within a loop construct.

```
*#           SELECT :PART-NAME = P.NAME, :PART-COLOR = P.COLOR
*#               FROM PARTS P
*#           {
*#               BEGIN TRANSACTION;
*#               INSERT INTO COLORTABLE (CNAME CCOLOR)
*#                   VALUES (:PART-NAME :PART-COLOR);

                 IF NDML-STATUS = 'ERROR'
*#                   ROLLBACK;
                 ELSE
*#                   COMMIT;
                     .
*#           }
```

Other examples are shown in the Appendices A and B.

SECTION 4

NDML PROCESSING

4.1 IISS Precompiler Overview

The IISS Precompiler will precompile a user's application process containing embedded NDML commands. The Precompiler parses the application program source code and identifies the NDML commands. It will modify the original application process to include numerous variables and subroutine calls necessary to implement the NDML commands in the host language. The Precompiler will generate code (generated query processes) that will be activated at run time to access the identified internal-schema databases and to perform the required internal-schema to conceptual-schema transforms. It will also generate code (generated conceptual/external transformer) that will be activated at run time to perform the required conceptual to external transforms, statistics functions, ordering of results, and other processes necessary to present the requested results to the application process.

In order to execute the IISS Test Bed Precompiler, the user must enter the IISS Test Bed System. After entering his username, password and role he will then be shown the CHOOSE FUNCTION form. The function name CDCDPREZZZ should be entered to start the Precompiler. The next form shown will be the PRECOMPILER INPUT form. The user should enter the application process name, the target host of the application process, the source file name and the source listing file. The Precompiler will execute and the user will be returned to the CHOOSE FUNCTION form. Following is an example of the forms and the required responses to execute the IISS Test Bed Precompiler. At end of precompilation, an error count message will appear at the bottom of the function screen.

After precompilation is complete, leave UIMS and return to VAX/VMS. Review the source listing file using $EDT, $TYPE or $PRINT to check for errors and instructions concerning generated code which must be compiled, linked, tested, and added to the NTM tables and possibly transferred to a different host. Note that the NTM requires that all file names be 8 characters; consequently, names in the generated code will be padded with the character Z to achieve the required length. At this writing, a complete list of Precompiler-generated error messages is not available.

Changes can be made to the original source code with an editor if need be and the program precompiled again. The Precompiler will automatically delete generated Application Process name references from the data base, but will not delete out of date versions of source code, objects and executable images.

4.2 Example of Precompiling and Executing Application Process

Following is an example of precompiling and executing an Application Process. The example will be using Application Process CDTS1. This is a COBOL Application Process; however, the steps detailed below are the same whether the users application is a COBOL or FORTRAN program.

Step 1 - Execute the IISS Precompiler

Enter the IISS Test Bed System. After entering username, password and role, the CHOOSE FUNCTION form will be displayed. At this time CDCDPREZZZ is entered as the function to start the Precompiler (see Figure 4-1). Figure 4-2 shows the responses required to precompile Application Process CDTS1. The TARGET HOST is the name of any valid host within the IISS system. The SOURCE FILE is the file name for the source code before precompilation and SOURCE LISTING FILE is the file name to contain the source code after precompilation (these two file names must be different). Figure 4-3 shows the corresponding responses if CDTS1 were a FORTRAN Application Process. At the completion of precompilation the CHOOSE FUNCTION form will be displayed and at this time EXIT should be entered.

```
                 IISS   TEST   BED   VERSION   1.0
--------------------------------------------------------------

DATE:   /  /   TIME:   :   :   USER ID:         ROLE:
     -- -- --       --  --  --         -------        -------
FUNCTION:CDCDPREZZZ
         ----------
```

Figure 4-1. Execute Precompiler

```
                         IISS TESTBED PRECOMPILER

PLEASE ENTER THE FOLLOWING INFORMATION ABOUT THE APPLICATION
PROCESS TO BE PRECOMPILED:

     APPLICATION PROCESS NAME                    - CDTS1

     TARGET HOST OF APPLICATION PROCESS          - VAX

     SOURCE FILE NAME                            - SNGTEST.PRC

     SOURCE LISTING FILE                         - CDTS1.COB
```

Figure 4-2. Precompiler Application Process Responses
(COBOL)

```
                    IISS TESTBED PRECOMPILER

PLEASE ENTER THE FOLLOWING INFORMATION ABOUT THE APPLICATION
PROCESS TO BE PRECOMPILED:

     APPLICATION PROCESS NAME                 - CDTS1

     TARGET HOST OF APPLICATION PROCESS       - VAX

     SOURCE FILE NAME                         - SNGTEST.PRC

     SOURCE LISTING FILE                      - CDTS1.FOR
```

Figure 4-3. Precompiler Application Process Responses (FORTRAN)

Step 2 - Review the Precompiled Application Process Source Listing File

Edit the source listing file, for this example CDTS1.COB, and locate the text string "ERROR LISTING". Print the text starting at this line until the end of the file. Figure 4-4 shows an example of the output for the CDTS1 Application Process.

File B0008.TMP is the error listing file generated during the precompilation of the Application Process. FILE A0005ZZZ.COB is the generated Conceptual/External Transform Program. File A0006ZZ.COB is the generated Request Processor. There will always be one error file but could be many C/E Transform Programs and Request Processors depending on the number and complexity of the NDML queries in the Application Process. The HOST column indicates the computer host where the code must be transferred if different from the VAX. The TYPE column indicates the type of generated application process, either a conceptual/external transformer or a query processor. The A*.COB and B*.TMP will be different for each run of the Precompiler.

Step 3 - Review the Error Listing File

Review the error listing file using $EDT, $TYPE or $PRINT to check for errors that might have occurred during the precompilation of the Application Process. If errors have been detected, change the original source code (SNGTEST.PRC) and precompile the Application Process again starting at STEP 1, otherwise continue with Step 4.

Step 4 - Compile the Precompiled Application Process and the Generated Application Processes.

The precompiled Application Process and the NDML programs generated during the precompilation phase must be compiled in order to create object files for each program. The programs generated during the precompilation phase to implement NDML commands will be Conceptual/External Transformers and Request Processors. The number of these NDMP programs will depend on the complexity of the NDML query in the original users

```
*ERROR LISTING IS B0008.TMP
*
*THE GENERATED APPLICATION PROCESSES FOR THIS PROGRAM ARE:
*
*    FILE NAME HOST  TYPE
*    A0005ZZZ.COB    VAX     CONCEPTUAL/EXTERNAL TRANSFORMER
*    A0006ZZZ.COB    VAX     ORACLE QUERY PROCESSOR
```

Figure 4-4. Example Output

application process. The following procedure files have been provided to compile any program:

COMPANS	- compile a COBOL program/subprogram
FORANS	- compile a FORTRAN program/subprogram

The parameters for the compile procedure are as follows:

```
COMPANS P1
FORANS P1
```

where P1 = file name of the program

For this example three programs must be compiled as shown below:

```
$@COMPANS CDTS1
$@COMPANS A0005ZZZ
$@COMPANS A0006ZZZ
```

Step 5 - Link the Precompiled Application Process and the Generated NDML processes

The programs that were compiled in Step 4 must now be linked in order to create an executable file for each program. The link procedure file for the query processors will vary depending on the data base manager that must be accessed in order to satisfy the NDML query. The following procedure files have been provided to link any programs/subprograms.

LNKAP	- link a precompiled Application Process
LNKCE	- link the Conceptual/External Transformer
LNKQERY	- link an ORACLE Request Processor
LNKSUB	- link a precompiled Application Process and associated subprograms

The parameters for the link procedures are as follows:

```
LNKAP    P1  P2
LNKCE    P1  P2
LNKQERY  P1  P2
```

where

P1 = name of the program
P2 = link parameter (i.e. DEBUG, NODEBUG, NOMAP)

```
LNKSUB  P1  P2  P3
```

where

P1 = name of the main program
P2 = name of the subprogram
P3 = link parameter (i.e. DEBUG, NODEBUG, NOMAP)

For this example, three programs must be linked as shown below:

```
$@LNKAP       CDTS1   NODEBUG
$@LNKQERY     A0006   NODEBUG
$@LNKCE       A0005   NODEBUG
```

NOTE: If the generated code must be transferred to another host, one must first transfer the code and then perform Steps 4 and 5 on that host, using procedures to be determined. Other link processors will be required for other dbms-host combinations.

Step 6 - Add the Application Process to the NTM Tables

If this is the first time the Application Process has been precompiled some NTM tables must be changed in order to reflect the new Application Process and its generated NDML Processes. The modification of these tables involves editing a table initialization file to make the necessary changes. The structure of the tables and an example of an initialization data record is included in the IISS Test Bed Network Transaction Manager Operator's Manual. The following NTM tables must be modified when adding a new Application Process:

ACTTBL	=	Authority Check Table
APITBL	=	Application Process Information Table
APTTBL	=	Application Process Characteristics Table

Step 7 - Add the AP/ROLE Relation to the UI Data Base

If this is the first time the Application Process has been precompiled, a row must be added to the AP/ROLE relation found in the UI data base. This consists of executing ORACLE and using the ORACLE UFI language to add the new AP/ROLE data.

Step 8 - Define the Application Process to the UIMS

If this is the first time the Application Process has been precompiled, the Application Process must be defined to the UIMS. Procedures to define the Application Process can be found in the IISS User Interface Management System Services User Manual.

Step 9 - Execute the Precompiled Application Process

Application Process, one must enter the IISS Test Bed System as described in Step 1. At the CHOOSE FUNCTION form, enter the name of the Application Process, in this case CDCDTS1ZZZ (see Figure 4-5). The Application Process will now be executed. Application Process CDTS1 does not use the forms interface for accepting inputs or displaying results; therefore, when executing CDTS1, all prompts and results will be displayed on the operators console. In order to receive these prompts and results one must use the following VAX/VMS ASSIGN statement for the operators console:

```
$ ASSIGN/GROUP   _TTnn: SYS$COMMAND
```

where nn is the process number for the current terminal determined by issuing the VAX/VMS SHOW PROCESS command.

NOTE: Steps 6-9 are used only for the precompiled Application Process.

```
                    IISS TEST BED VERSION 1.0
---------------------------------------------------------------
DATE:  /  /  TIME:   :  :   USER ID:          ROLE:
     -- -- --      -- -- --          -------        -------
FUNCTION:  CDCDTS1ZZZ
           ------------
```

Figure 4-5. Execute Precompiled Application

4.3 COBOL Reserved Names and Labels

During the precompilation phase of IISS, code is generated into the generated application process source code. For a COBOL application process, code is generated in the File Control Section, File Section, Working-Storage Section and Procedure Division. Following is a list of reserved file names, variable names and label statements for any COBOL application process used in the IISS environment. These are legal COBOL variables and statements but must not appear in the original application process as user defined variables or labels.

(a) File Names

RESULT - nn

where nn	depends on the nesting structure of the embedded NDML query statements

(b) Variable Names

CDM-**
RES-**
SS-**
NDML-STATUS
NDML-COUNT

where **	suffix added depending on the embedded NDML query statements

(c) Label Statements

CDM-LOOP-nn
CDM-EXIT-nn
CDM-ESCAPE-nn

where nn	depende on the nesting structure of the embedded NDML query statements

4.4 FORTRAN Reserved Names and Labels

During the precompilation phase of IISS, code is generated into the original users application process source code. This generated code consists of variable definition statements and formatted input/output statements. Following is a list of reserved variable names, CONTINUE statements, FORMAT statements and logical unit numbers for any FORTRAN application process used in the IISS environment. These are legal FORTRAN variables and statements but must not appear in the original application process as user defined variables or labels.

(a) Variable Names

```
APHOST
CDCHAN
CDCSRT
CDDLEN
CDDTYP
CDMSNV
CDMSRC
CDMTYP
CDPTR
CDTVAL
CDWFLG
DECMAL
NTMSTA
NSTATS
NCOUNT

ACnnnn
CHnnnn
DInnnn
Fnnnnnn
MSGInn
MSGOnn
RSnnnn
```

where nn...   depends on the nesting structure of the embedded NDML query statements

(b) CONTINUE Statements

```
997nn - 999nn   CONTINUE
```

where nn depends on the nesting structure of the embedded NDML query statements

(c) FORMAT Label

```
996nn   FORMAT
```

where nn depends on the nesting structure of the embedded NDML query statements

(d) Logical Unit Numbers

The logical unit numbers for the result files will be numbered starting at 99 and decremented by one for each NDML query statement.

APPENDIX A

BNF OF THE NDML

A.1 Conventions

Certain conventions are used to describe the form of commands:

UPPER CASE WORDS denote keywords in the command

LOWER CASE WORDS denote user-defined words

{ } denotes that exactly one of the options within the braces must be selected by the user

"{" or "}" denotes a literal brace character without special meaning

[ ] denotes that the entry within the brackets is optional

| denotes an "or" relationship among the entries

_ denotes default option

The only punctuation allowed is:

(1) a "." to separate the table-label (.e., table alias) from the column-name. The table-label is used to match a column to a specific table in the list of tables referenced in the FROM clause,

(2) a ":" before the name of a host-language program variable, structure or file name that will receive returned values,

(3) a "," between entries in the list of tables in a FROM clause,

(4) a "," between subscripts to an array variable,

(5) a set of parentheses to enclose the column-list in an INSERT statement,

(6) a set of parentheses to enclose the object column of a function,

(7) a set of parentheses to enclose the values to be inserted in an INSERT statement,

(8) a set of parentheses to enclose a program variable subscript list,

(9) a mandatory ";" or loop-construct at the end of the command.

Only upper-case letters are recognized by the NDML Precompiler.

A.2 NDML Backus-Normal Form (BNF)

```
ndml-command          ::= select-command | insert-command |
                          delete-command | modify-command |
                          begin-recoverable-unit-command |
                          commit-command | rollback-command

select-command        ::= SELECT [lock-request]
                             [INTO external-struct] [DISTINCT]
                             {[table-label] ALL | expr-list |
                                var-assgnmt-list}
                          FROM table-list
                          [WHERE predicate-list]
                          [ORDER BY order-spec-list]
                          {; | loop-construct }

insert-command        ::= INSERT INTO table-name
                             (column-list)
                          VALUES {FROM external-struct |
                             source-list};

delete-command        ::= DELETE FROM table-name
                             [table-label]
                          [USING table-list]
                          WHERE {ALL | predicate-list};

modify-command        ::= MODIFY table-name [table-label]
                          [USING table-list]
                          SET column-assgnmt-list
                          WHERE {ALL | predicate-list};

begin-recoverable
     unit-command       ::= BEGIN TRANSACTION;
```

```
commit-command          ::= COMMIT;

rollback-command        ::= UNDO; | ROLLBACK;

bool-op                 ::= = | != | > | >= | < | <=

column-assgnmt-list     ::= column-assgnmt-spec |
                               column-assgnmt-list
                               column-assgnmt-spec

column-assgnmt-spec     ::= column-spec = value

column-list             ::= column-spec | column-list column-spec

column-predicate        ::= column-spec bool-op value | value
                               bool-op column-spec

column-spec             ::= column-name | table-name.column-
                               name | table-label.column-name

digit                   ::= 0|1|2|3|4|5|6|7|8|9

direction               ::= ASC | DESC | ASCENDING |
                            ---
                               DESCENDING | UP | DOWN

expr-list               ::= expr-spec | expr-list expr-spec

expr-spec               ::= column-spec | function([DISTINCT]
                               column-spec)

external-struct         ::= FILE 'file-name' |
                               FILE ':variable-name' |
                               STRUCTURE :variable-name

function                ::= AVG | MEAN | MAX | MIN | SUM | COUNT

integer                 ::= digit | integer digit

join-op                 ::= = | !=

join-predicate          ::= column-spec join-op column-spec

lock-request            ::= WITH [EXCLUSIVE | SHARED | NO] LOCK
                                                          --

loop-construct          ::= "{" statement-list "}"
```

```
number              ::= integer [.[integer]]

order-spec-list     ::= column-spec [direction] |
                           order-spec-list column-spec
                           [direction]

predicate-list      ::= predicate-spec | predicate-list AND
                           predicate-spec

predicate-spec      ::= column-predicate | join-predicate

quoted-variable     ::= 'literal-string'

scalar-variable     ::= :variable-name [(subscript-list)]

source-list         ::= (value-list)

statement           ::= host-language-statement |
                           ndml-command | BREAK | EXIT |
                           CONTINUE | NEXT

statement-list      ::= statement |
                           statement-list statement

subscript-list      ::= integer | subscript-list , integer

table-list          ::= table-name [table-label] | table-
                           list, table-name [table-label]

value               ::= scalar-variable | quoted-variable |
                           number

value-list          ::= value | value-list value

var-assgnmt-list    ::= var-assgnmt-spec |
                           var-assgnmt-list var-assgnmt-spec

var-assgnmt-spec    ::= scalar-variable = expr-spec
```

APPENDIX B

COBOL EXAMPLE PROGRAM

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. VOMAPS.
*
*******************************************************************
*
*      PROGRAM NAME     VOMAPS
*
*
*******************************************************************
*
* DESCRIPTION :  THIS ROUTINE ENFORCES THE FOLLOWING AUC
*                TO SET MAPPING RULES:
*                  1.  AN AUC ALWAYS MAPS TO THE SAME DATABASE
*                  2.  THE AUC VALUE MUST BE UNIQUE.
*                  3.  THE SET OWNERS RECORD TYPES MUST BE
*                      BE IDENTICAL.
*                      IF NO RULES ARE BROKEN, THE RETURN CODE
*                      IS 1, OTHERWISE ZERO.
*
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-11.
 OBJECT-COMPUTER. VAX-11.
*
 DATA DIVISION.
*
* DATA ORGANIZATION :
*
 WORKING-STORAGE SECTION.
*
* INCLUDE FILES
*
 COPY SRVRET OF IISSCLIB.
```

```
*
* LOCAL VARIABLES
*
01  MODULE-NAME                 PIC X(6) VALUE "VOMAPS".
01  MESG-DESC                   PIC X(60).
01  RET-STATUS                  PIC X(5).
01  DB-NO                       PIC S9(9) COMP.
01  SET-NAME                    PIC X(30).
01  REC-NAME                    PIC X(30).
01  AUC-VALUE                   PIC X(30).
*
* INTERFACES :
*
 LINKAGE SECTION.
*
*
* INPUT ARGUMENTS
*
 01  IN-DB-NO                   PIC S9(9) COMP.
 01  IN-SET-NAME                PIC X(30).
 01  IN-OWNER-REC-NAME          PIC X(30).
 01  IN-TAG-NO                  PIC S9(9) COMP.
 01  IN-AUC-VALUE               PIC X(30).
*
* OUTPUT ARGUMENTS
 01  OUT-CODE                   PIC S9(9) COMP.
*
*
* LIMITATIONS :
*
*
* PROCESS DESCRIPTION :
*     PERFORM AN NDML SELECT TO OBTAIN INFORMATION ON
*     PREVIOUS MAPPINGS FROM THE TAG NUMBER.  FOR EACH
*     ROW SELECTED, ENSURE THAT THE ABOVE 4 RULES ARE
*     ENFORCED.  IF NOT, GENERATE AN APPROPRIATE ERROR
*     MESSAGE AND RETURN WITH OUT-CODE EQUAL ZERO,
*     OTHERWISE OUT-CODE EQUALS ONE.
*
 PROCEDURE DIVISION USING    IN-DB-NO
                             IN-SET-NAME,
                             IN-OWNER-REC-NAME
                             IN-TAG-NO
                             IN-AUC-VALUE
                             OUT-CODE.
```

```
*
 START-PROGRAM.
     MOVE 1 TO OUT-CODE.
*#   SELECT :DB-NO = A.DB_ID,
*#           :SET-NAME = A.SET_ID,
*#           :REC-NAME = B.RT_ID_OF_OWNER,
*#           :AUC-VALUE = A.AUC_VALUE
*#   FROM   AUC_ST_MAPPING A,
*#          RECORD_SET B
*#   WHERE  A.TAG_NO = :IN-TAG-NO AND
*#           A.DB_ID = B.DB_ID AND
*#           A.SET_ID = B.SET_ID
*
*#   {
*
*    APPLY RULE 1 - THE DATABASE FROM THE SELECT
     STATEMENT MUST MATCH THE USER ENTERED DATABASE
*
     IF DB-NO NOT = IN-DB-NO
         MOVE 0 TO OUT-CODE
         MOVE "AUC MAY NOT BE MAPPED TO DIFFERENT DATABASES"
              TO  MESG-DESC
         CALL "UERROR" USING MESG-DESC
*#   EXIT
     .
*
*    APPLY RULE 2 - THE AUC VALUE MUST BE UNIQUE
*
     IF AUC-VALUE = IN-AUC-VALUE
         MOVE 0 TO OUT-CODE
         MOVE "THE FOLLOWING AUC VALUE IS NOT UNIQUE - "
             TO MESG-DESC
         CALL "UERROR" USING MESG-DESC
         MOVE IN-AUC-VALUE TO MESG-DESC
         CALL "UERROR" USING MESG-DESC
*#   EXIT
     .
*
*    APPLY RULE 3 - THE SET OWNERS RECORD TYPES MUST MATCH
*
     IF REC-NAME NOT = IN-OWNER-REC-NAME
         MOVE 0 TO OUT-CODE
         MOVE "NON MATCHING SET OWNERS RECORD TYPES"
             TO MESG-DESC
         CALL "UERROR" USING MESG-DESC
*#   EXIT
     .
*
*#   }
     IF NOT OK
         GO TO NDML-ERROR.
 AFTER-LOOP.
*
*
*
* RETURN
*
 EXIT-PROGRAM.
*
     EXIT PROGRAM.
*
 NDML-ERROR.
     MOVE NDML-STATUS TO RET-STATUS.
     MOVE 'NDML ERROR TRAPPED' TO MESG-DESC.
     PERFORM PROCESS-ERROR.
     GO TO AFTER-LOOP.
*
*
* PROCESS ERROR
*
COPY ERRPRO OF IISSCLIB.
/
```

APPENDIX C

FORTRAN EXAMPLE PROGRAM

```
      PROGRAM CDTS8
C
C    APPLICATION PROCESS CDTS8
C
C  AN EXAMPLE OF AN APPLICATION PROCESS
C  CONTAINING A NESTED NDML QUERY.
C
      CHARACTER ECNAME*30,AUCNAM*30
      CHARACTER V1*30,V4*30
      INTEGER V5,V6,V8
      INTEGER V2,V3,V7,NTMBSZ
      DATA NTMBSZ /4096/
      CHARACTER NTMBUF*4096,TERMID*2,NTMSTS*1,RETCOD*5
C
C .............................
C
      CALL INITAL (%REF(NTMBUF),
     *             %REF(NTMBSZ),
     *             %REF(SYSSTS),
     *             %REF(RETCOD))
      IF (RETCOD .NE. '00000') THEN
         PRINT *,'BAD INITAL'
         CALL TRMNAT (%REF(NTMSTS))
      ENDIF
  100 CONTINUE
         PRINT *, 'ENTER ENTITY CLASS LABEL'
         READ (UNIT=5,FMT=200) ECNAME
 200     FORMAT (A20)
      IF (ECNAME .EQ. 'EXIT')  GO TO 900
C
C#        SELECT :V1 = R.EC_LABEL :V2 = R.EC_NO
C#               :V3 = R.DEP_EC_NO :V4 = R.RC_LABEL
C#           FROM RELATION_XREF R
C#           WHERE R.EC_LABEL = :ECNAME
C#     {
           PRINT *,'ENTITY CLASS LABEL      :',V1
           PRINT *,'ENTITY CLASS NUMBER     :',V2
           PRINT *,'DEPENDENT ENTITY CLASS  :',V3
           PRINT *,'RELATION CLASS LABEL    :',V4
           PRINT *,' '
C#      SELECT :V1 = I.EC_LABEL :V2 = I.EC_NO
C#             :V5 = I.TAG_NO :V6 = I.TAG_INHER
C#             :V7 = I.IND_EC_NO
C#         FROM  IAUC_XREF I
C#         WHERE     I.IND_EC_NO = :V2
C#              AND I.EC_NO      = :V3
C#              AND I.RC_LABEL   = :V4
C#      {
           PRINT *,'    DEPENDENT ENTITY CLASS LABEL:',V1
           PRINT *,'    DEPENDENT ENTITY CLASS NUMBER :',V2
           PRINT *,'    ATTRIBUTE USE CLASS NUMBER    :',V5
           PRINT *,'    INHERITED FROM AUC NUMBER     :',V6
           PRINT *,'    INHERITED FROM EC NUMBER      :',V7
           PRINT *,' '
C#      }
C#     }
      GO TO 100
C
  900 CONTINUE
      CALL TRMNAT (%REF(NTMSTS))
      END
```

APPENDIX D

REFERENCES

Related ICAM Documents included:

UM62014100	CDM Administrator's Manual
TBM620141000	CDM1, An IDEF1 Model of the Common Data Model
UM620141002	Information Modeling Manual - IDEF1-Extended (IDEF1X)
UM620141100	Neutral Data Definition Language (NDDL) User's Guide
DS620141200	Development Specification for the IISS NDML Precompiler Configuration Item
DS620141320	Development Specification for the IISS Aggregator Configuration Item
DS620141310	Development Specification for the IISS Distributed Request Supervisor Configuration Item

U.S.Government Printing Office 1987 748-061/60886

END
7-87
DTIC